# Maps demo

Note: before re-running the demo:

* run cleanup.sh script
* delete the githubMysql broker
* delete GithubStats V2 Api descriptor
* delete Githubstats V2 Api descriptor

This needs to be done only once in the cluster, If githubstats Broker does not exist:

* Go to the Microservice management console
* Select ‘New User Service’
* Use the githubstats.tar.gz file under the brokers directory to create the broker.

This needs to be done only once in the cluster:

kubectl create -f deployments/githubstats-ingress.yaml

kubectl create -f deployments/map-ingress.yaml

kubectl create -f deployments/githubstats-service.yaml

Demo steps:

1. I’m the backend developer, I have the code for a service that pull and aggregate data from github.
2. I have a docker image already deployed in a registry so now I want to deploy the service in the cluster

kubectl create -f deployments/githubstats-v2.yaml

1. Show that the service is starting with

kubectl get po

1. I want to make easier for the developer(s) that want to use my service so now I want to publish an ApiDescriptor that specifies the APIs that my service exposes
2. Go to the Microservices admin console
3. Go to APIs
4. Create a new API by uploading the blueprint file under apis/githubstatsv2.md
5. After the APi file is uploaded, bind it to a service instance, in the services’ dropdown select githubstats V2
6. Now that I have an ApiDescriptor bound to a service I can move to the frontend service
7. Go to the Microservices admin console, select the GithubStats API V2
8. Select ‘View API’, show the description of the APIs and that there is the possibility to use a mock service to test the Apis if the service behind them is not deployed.
9. Select ‘download client library’ (Javascript)
10. You can unzip the library and open the frontend project in an editor (ex. VS Code) but you can skip it and say that this library allows the frontend client to connect to the backend without having to find out where the service is running (IP/Port number) and having to deal with making the HTTP REST calls, the library takes care of all of that
11. After the frontend service has been updated, it is packaged in a docker image and pushed to a registry (the service has been already published to save time so no need to actually run through the process)
12. Now I want to deploy the frontend service:

kubectl create -f deployments/map-deployment-v2.yaml

1. I also need to create a service to expose my frontend service so

kubectl create -f deployments/map-service.yaml

1. Now I can go to the IP address of the cluster and see the service running by going to the following url <cluster IP>/map/
   1. Note: you can find the IP address to use by running this command:

kubectl get ing

1. Now we want to make the backend service better because our customers asked us to provide a way to go back in-time and see how the map change so we need to store the historical data in a database (mysql in this case)
2. First I create a Broker that will manage the connection to the database
3. Go to the microservices admin console
4. Create a new ‘User Service’ using the mysqlbroker.tar.gz file under the /brokers directory
5. Now you can imagine that I updated my service and published the image on a repo
6. I want to deploy the new version of the service now

kubectl create -f deployments/githubstats-v3.yaml

1. There are two version of my service now running in the cluster, show it with

kubectl get po -l app=githubstats

1. Now, like I have done before, I publish a new version of my APIs (the new service exposes new APIs to query historical data
2. Goto the microservice admin console, repeat the process and create a new Api by uploading apis/githubstatsv3.md
   1. Note: there is no need to bind a service to the description
3. Now I can go back to the frontend service and update it, like I have done before, I update the code and publish a new version of my docker image
4. I deploy the new version in the cluster:

kubectl create -f deployments/map-deployment-v3.yaml

1. I have two versions of the frontend and two versions of the backend, show it via

Backend:

kubectl get po -l app=githubstats

Frontend:

kubectl get po -l app=activitymap

1. I want now to redirect traffic to the right versions using the service mash
2. First I redirect traffic to the V3 version of my backend service: istioctl replace -f mesh/githubstats-routing-v2.yaml
   1. Now all traffic to the Githubstats service goes to the V3 version
3. I want to show the new frontend, so like I have done with the backend service, I redirect traffic to the new version: istioctl replace -f mesh/map-v3-routing.yaml
4. Go to the maps page and how I should see the slides on the side that allows to move in the timescale
5. Now the service is running so I want to be able to monitor it
6. Before going to the grafana/zipkin console, have 3 bash shells open and run the 3 bash scripts under the /load directory, one on each shell
   1. Note: you have to have ‘watch’ installed so brew install watch
   2. Note: you have to edit the bash scripts and update the IP address in the curl commands (will make it a parameter in the future)
7. After all 3 scripts are running you can go in the grafana dashboard and so that the system is collecting metrics about the running services
8. You can move to the zipkin console and show that it shows, in realtime, data flowing across services. You can select one and show that the UI displays the two versions of the (for instance) githubstats service and you can tell that, in the future, we will be able to show information about the actual APIs that are used.